NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPROVED NETWORK SECURITY AND DISGUISING TCP/IP FINGERPRINT THROUGH DYNAMIC STACK MODIFICATION

by

Aaron C. Judd

September 2005

Thesis Advisor:	James Bret Michael
Second Reader:	Man-Tak Shing

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Improved Network Security and Disguising TCP/IP Fingerprint Through Dynamic Stack Modification	**5. FUNDING NUMBERS**
6. AUTHOR(S) Aaron C. Judd	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Space and Naval Warfare Systems Center San Diego, CA	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

Each computer on a network has an OS Fingerprint that can be collected through various applications. Because of the complexity of network systems, vulnerabilities and exploitations of the same to gain access to systems will always be a problem. Those wishing to attack a system can use the OS Fingerprint to identify the types of vulnerabilities and software exploits that will be effective against the system. This paper discusses how system vulnerabilities become exploited and used by network attackers. Because OS Fingerprints are one of many tools network attackers will use to identify and attack a system, concealing a system's OS Fingerprint becomes an important part of securing that system. To demonstrate the capability of concealing the OS Fingerprint of a system, a prototype system was developed. This prototype changed the OS Fingerprint of a Linux system so that it matched a Windows NT system.

14. SUBJECT TERMS Network Security, TCP/IP, OS Fingerprinting	**15. NUMBER OF PAGES** 59
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPROVED NETWORK SECURITY AND DISGUISING TCP/IP FINGERPRINT THROUGH DYNAMIC STACK MODIFICATION

Aaron C. Judd
Civilian, SSC-SD, Code 246212
B.S., Computer Science, Brigham Young University, 1997

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2005

Author: Aaron C. Judd

Approved by: Dr. James Bret Michael
Thesis Advisor

Dr. Man-Tak Shing
Second Reader

Dr. Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Each computer on a network has an OS Fingerprint that can be collected through various applications. Because of the complexity of network systems, vulnerabilities and exploitations of the same to gain access to systems will always be a problem. Those wishing to attack a system can use the OS Fingerprint to identify the types of vulnerabilities and software exploits that will be effective against the system. This paper discusses how system vulnerabilities become exploited and used by network attackers. Because OS Fingerprints are one of many tools network attackers will use to identify and attack a system, concealing a system's OS Fingerprint becomes an important part of securing that system. To demonstrate the capability of concealing the OS Fingerprint of a system, a prototype system was developed. This prototype changed the OS Fingerprint of a Linux system so that it matched a Windows NT system.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

Donna Judd

My beautiful wife, without whose encouragement, I would never have finished.

Dr. Bret Michael

For his rescue of this work from email when all was lost during a hard drive crash.

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This paper discusses how system vulnerabilities become exploited and used by network attackers. Because OS Fingerprints are one of many tools network attackers will use to identify and attack a system, concealing a system's OS Fingerprint becomes an important part of securing that system. To demonstrate the capability of concealing the OS Fingerprint of a system, a prototype system was developed. This prototype changed the OS Fingerprint of a Linux system so that it matched a Windows NT system. This effort found that OS Fingerprints can be changed by making configuration changes to default OS kernel software and by compiling a custom OS kernel with modified source code. OS Fingerprints may change as more varied systems and hardware are available and used. Managing the OS Fingerprint of a network system is an additional way to help protect the system from attack and should be considered as part of a comprehensive approach to network security.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. NETWORK SECURITY AND ATTACK PROFILE

There is no such thing as perfect security. A computer that is not even connected to a network is still subject to risk from an operator with access to the console. Once the system is attached to a network, the number of possible access sites grows to include every computer in that network. Computer systems use multiple levels of software to abstract the complex operations of the computer such as video output, keyboard input, network and hard drive access. These hardware abstractions are managed by the software in the Operating System (OS). The OS uses hardware drivers to control the various hardware components. The primary function of the OS is to control access to these varied hardware components and avoid conflicts between users and applications. The OS also has Application Programming Interface (API) that other software applications use to access the hardware.

Because of these layers of abstraction, there is also a need for layers of security. A software application can be more secure. But once an attacker has gained access to the OS, he can then take control of the application. If the OS is more secure, an attacker can take control of a trusted application to gain access to parts of the computer that the OS is trying to secure.

Network security is an ongoing process that requires the ever vigilant attention of an experienced system administrator. Each level of abstraction in the computer system needs to be maintained for security [1, 2, 3]. With the ever growing complexity and size of systems and applications, new vulnerabilities emerge. It is possible to eliminate some of these vulnerabilities by modifying the software, or through other configurations or restrictions to limit access to unauthorized users or applications. Every computer system relies on hundreds of applications, components, libraries, drivers, files and configurations. All of which can become a target once a vulnerability is found. New methods of attacking a computer system are being created each day, and as new vulnerabilities are being discovered in existing software systems. Even when software systems are modified to eliminate a known vulnerability, the modifications can potentially introduce new vulnerabilities. Thus the never ending search for security will continue.

Computer systems with vulnerabilities to a network attack can be exploited by any computer with access to the network. Protecting the system from network attacks becomes important. Some methods used to protect computer systems from network-based attacks include: limiting the number and type of network services being provided, restricting access to specific computers and services with a firewall, restricting access to specific user accounts, keeping OS and network software patched and updated. These restrictions can help protect a computer system on the network, but at the cost of limiting operations to a defined set of trusted users and network systems. In addition, there may be new vulnerabilities that will be discovered. There may be new applications that are not yet installed that can introduce new types of vulnerabilities and may not be compatible with this restrictive firewall solution. Once an attacker accesses the network beyond the firewall, the protection it provided can be circumnavigated.

There are essentially two different types of network attacks, novice and expert attackers. The novice attacker, also known as "Script Kiddies," is able to attack a system using programs and libraries created by other more skilled attackers. Because these types of attacks do not require advanced understanding of computers, these attacks are the most common. Fortunately the vulnerabilities that these novice attackers apply tend to become well known, with patches and configuration changes becoming available to nullify the effectiveness of the script-kiddy attack program. A novice attacker may not even know how the scripted attack works, or what vulnerability or system it will work against. These attackers succeed because so many users and maintainers of computer systems fail to patch even the most basic vulnerabilities in a timely manner. As with random violence, it is nearly impossible to avoid being targeted by attackers. Vigilant patching and configuring of the system needs to be a part of organizations' and individuals' approach to maintaining secure operation of computer systems. By attacking randomly computer systems that are not vulnerable to the specific attack, a novice attacker may be detected before targeting an actual vulnerable system. This can allow the vulnerable systems to be protected by blocking all access from the computer where the attacks are coming from.

At the other end of the spectrum are attackers who are highly skilled at developing software and have deep knowledge of the systems they are attacking; these people may even be professional information warriors, of sorts, sponsored by organized crime, terrorist groups, or governments (e.g., military or intelligence agencies). These attackers study the applications they are trying to attack. They will try to decompile the application back into its original source code, run the application in a virtual environment to observe its operation, study the database and file structures applications and system-level software use, and where possible, obtain source code and document of the application or system artifacts (e.g., requirements specifications, design documents, user manuals). All of these activities require strong understanding of the design and operation of the system across the many levels of abstraction. These attackers look for vulnerabilities that are not yet well known or possibly have not already been uncovered: these vulnerabilities provide the professional hacker with the opportunity to experiment and test a specific attack on the vulnerability. Newly discovered vulnerabilities can depend on multiple levels of system abstraction in their complexity. This means that the attack may only work on one specific OS and one specific software version. It is unusual for a vulnerability to be generic enough that it will apply to multiple applications and operating systems, but it could occur due to software side effects (i.e., unplanned or unforeseen interactions between units of software). A new vulnerability may be backward compatible, applying to previous versions of a specific application. This attacker can test this new vulnerability on a system that he controls until he is confident that it will work on a real target. Once the vulnerability is widely known a new version of the application will be released to patch the vulnerability and a newer version of the application will no longer be affected by the same attack.

As the new attack is being tested and developed it will usually only be used on a limited number of machines and used by a limited number of attackers. Later these attacks can be written into scripts that less-skilled users can add to their arsenal of cyberattack tools. A new exploit can remain unknown until revealed when the novice attacker uses the simplified tool or script. These less experienced hackers tend to be less careful, make more mistakes and leave evidence of their activities. By the time these scripts or tools are available; the new vulnerability could have been exploited for a long time and not been detected. These unknown exploits are the most dangerous to network systems since no defenses are even available and no patches have been written.

An experienced attacker can perfect the new attack incrementally. First he will try to test and use the attack in a limited way. This experienced attacker will carefully select a target computer. He will try to search the network to find a system that matches his attack requirements. The system must have the same OS and software version.   To determine this, the attacker will use tools to scan the network. By protecting a computer from successful network scans, the amount of intelligence a potential attacker can gather decreases and security of the network system increases.   Protecting a network system from network scans is an important step in securing a network system from information warfare [4].

B. NETWORK SCANNING

A network scan is the process of attempting to open network connections on network ports of a network host [5]. The process of scanning a network computer uses the same protocols and procedures to make a legitimate connection. The only way to block a network scan is to limit the hosts and systems that can connect to the system being protected.

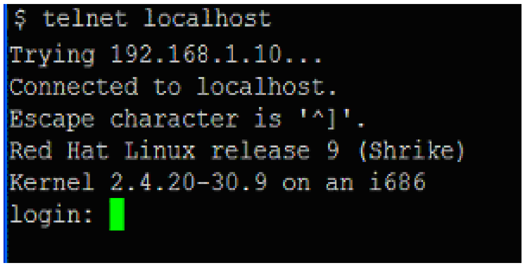
The first method of scanning a host is to send a PING request through TCP packet using ICMP [6]. ICMP is a protocol within the TCP protocol that allows for normal routing control. A PING packet will echo through the network and be replied by the host addressed in the packet. PING is a useful tool to determine if a machine is actively on the internet, disconnected or off. But it also allows an attacker to determine if the machine is on and thereby narrow his list of possible targets from all network addresses to only active hosts. Because of this firewalls can block ICMP packets to protected systems. The PING scan reveals only limited information, whether the network host is on or off, and is one of the least effective scans an attacker can use. The PING scan can be blocked by a firewall, but because detecting if a network host is active is a useful operation for normal network operation, this tends to degrade the normal operation of the network. Still, many network firewalls block PING packets to attempt to protect their systems from PING scans.

A PING scan cannot reveal many of the details an experienced attacker will need to know to craft an attack, such as which OS the system is running or what applications or services are available. Another type of network scan is a TCP port scan. This scan can reveal to the attacker which ports are open for connections and which are not. Normally applications create connections to ports that allow them to send and receive data. In a TCP port scan the normal network connection is limited to either a simple start up connection or a request to close the connection. Either can result in a response that will identify if the targeted port is open.

Each port has a specific number. Applications use “well known ports” to make finding applications on a specific machine possible without needing to attempt connections to all possible ports. Some examples of well known ports include port 80 for HTTP Web page services, port 25 for SMTP email services, port 21 for FTP, port 23 for TELNET console commands, port 22 for SSH secure console, port 110 for Pop3 email transport, and port 443 for HTTPS encrypted Web page services. There are up to 65,535 possible TCP ports, of which the first 2000 are allocated as “well known ports” for system services. An attacker can determine which services are running on the target host by looking at which well known ports are open.

At this point an attack can identify what hosts are active on the network (with a PING scan) and what services are available on that host (with a TCP port scan). The attacker still needs to identify which OS and applications are being used. It is possible to have several different vendors or applications that provide the same service. One machine might use window IIS for its Web server on port 80 while another would use Linux and Apache for the same Web server service. Each of these applications are created from different vendors and therefore do not normally have the same vulnerabilities. In the above example the Apache server has been ported to many different OSes, so even knowing which application is running does not necessarily reveal which OS it is running on. Each OS uses different directory structures for applications, different system call libraries, and different system accounts that need to be used for an successful attack.

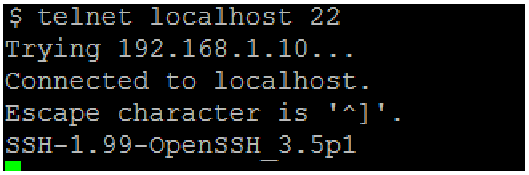
To determine what application and OS is running on the port, an attacker can try to grab a “Banner” or login message. Many services will identify themselves during normal communication. For instance a connection to a telnet server may reveal



```
$ telnet localhost
Trying 192.168.1.10...
Connected to localhost.
Escape character is '^]'.
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-30.9 on an i686
login:
```

Figure 1. Telnet Banner Capture

This banner message is created by the application and serves the purpose of helping other applications or operators to identify which system may offer specific methods. Sometimes the banner messages are necessary so applications can determine which version of the communication protocol to use.



```
$ telnet localhost 22
Trying 192.168.1.10...
Connected to localhost.
Escape character is '^]'.
SSH-1.99-OpenSSH_3.5p1
```

Figure 2. SSH Banner Capture

Some banner messages can be a gold mine to potential attackers. The above telnet banner reveals way too much information. Any network computer can connect without being challenged for a user login and can see the OS type (RedHat Linux) version (Shrike AKA RedHat 9.0) and kernel version (2.4.20-30.9) even the system architecture (i686). But banner messages are not fool proof. There is no guarantee that this information is accurate. A banner message can easily be modified to give no information or even false information.

Network scanning tools tend to have patterns in their search of the available ports of a target system that a human or tool can detect [7]. In an attempt to thwart network scans, tools have been written that can detect when a scan is taking place and from where it is coming. By detecting a network scan's unusual packets, ports accessed and packet order, a network scan detection tool can then block further network traffic coming from the known scanning computer. The detection capabilities have increased pressure on scanning tools and methods of scanning that are harder to detect such as FIN, stealth, slow and passive scanning.

C. OS FINGERPRINTING WITH TCP/IP

It is possible to use a network scan to determine the type of OS by identifying specific variations within the TCP/IP packet [8]. Each OS uses a different implementation of the TCP/IP protocol. The protocol was designed to be configurable and extendable. Since each OS can implement the TCP/IP protocol in slightly different ways, it is possible to determine which OS created the TCP/IP packet by looking at the details within the specific fields. While the TCP/IP fields do not actually carry a banner that identifies the OS, the variations are predictable, and by creating a library of OSes and their TCP/IP field variations it is possible to "fingerprint" an OS.

OS Fingerprints are a side effect of OS software design and engineering. When TCP/IP packets and connections are managed by the OS they are optimized and specialized for the particular hardware and configuration being used. The benefit of this optimization and customization may be out weighed by the risk of revealing details about the system to potential attackers on the network. As a Software System Engineering solution, the OS Fingerprint of a system should be carefully managed and controlled. A careful examination of the benefits of optimization and customization should be weighed against the risk of revealing unwanted information.

A TCP/IP fingerprint is a record of variations of the TCP/IP headers.   This fingerprint can verify an OS even with a falsified banner. The Nmap tool was the first tool to use this technology [9], and it is still the standard for TCP/IP port scanning and OS Fingerprinting. These variations result from the different way that each OS implements the TCP/IP protocols from the RFCs where they are specified.

1. Options

Because these protocols are designed to be modified and extended they were designed to be changeable with an option field. These changes are designed to be backwards compatible so those computers that don't implement the changes can still interoperate with computers that do. These changes or extensions help to improve the TCP/IP protocol performance or security. The TCP header sets aside a number of bits to identify which options are enabled. In some cases the OS will only implement part of the protocol and leave out optional features. Some of the TCP options that are commonly used in OS Fingerprinting are:

a. Timestamp

The timestamp option records the current time in a TCP packet that is sent over the internet. When this packet arrives at its target the time value is echoed back to the sender. This allows the sender to determine the round trip time for packets sent to that host and enables the host to optimize its TCP/IP protocol for maximum throughput, by allowing the remote computer to send multiple TCP packets while it waits for acknowledgements of their receipt.

b. Window Scale

This option is used to increase performance of high speed networks with relatively high latency by allowing the window size to scale above the normal 32,767 maximum by multiplying the advertised window size by the window scale value.

c. NOP (No Operation)

This option is used to add space to make the entire option field fit into a 32 bit word as required by the TCP protocol specification.

d. Maximum Segment Size (MSS)

MSS is used to specify the largest segment or packet that can be transmitted without requiring fragmentation. This value usually is set based on the type of network connection and media used to transfer data.

This value is determined by the media (i.e. twisted pair, fiber optic wire, wireless radio, etc) that is used to transport the data between computers. By supporting the MSS option the TCP/IP stack can agree on the segment size that is most efficient for communication over diverse networking media.

e. Order of Options and Spacing

In some cases the OS will simply select the ordering of fields differently but within the protocol. In the TCP protocol the options field doesn’t specify an order for the options only that they must fit into a full 32 bit word. This gives the OS writers great flexibility in how to implement these options. Some OSes use NOPs after each option to keep the total field length at 32 bits, while others always end with the Timestamp option. But this variation can be specific. (i.e. NetBSD will always have the Window Scale option before the Time Stamp option, where Linux will always have the Window Scale option after the Time Stamp option).

2. Implementation

In some cases the OS fails to correctly implement the protocol and errors are detectable, while not necessarily disrupting the capability of the TCP/IP protocol to work [10]. While these errors don’t tend to disrupt the normal use of the system, they can lead to unexpected behavior. These errors are especially visible when one type of OS is connecting to a computer using another. If both OSes don’t correctly implement the protocol, errors can occur and performance can suffer. Some examples of these include incorrectly responding to a FIN/ACK packet or accepting options in packets other than in the SYN packet. These options are only allowed in the setup portion of the TCP/IP protocol, but some OSes accept them in other packet messages. Sometimes these discrepancies between the OS implementation of the TCP/IP protocol and the correct implementation can result in errors that can crash the system as a result of certain types of TCP/IP packets received. The windows “SMURF” DOS (Denial of Service) attack was one of these. It used an invalid fragment of a packet that overloaded the windows network stack and crashed the entire system [11, 12].

a. UDP Port Closed

Some OSes incorrectly send a UDP port closed message to respond to invalid UDP requests, while others echo incorrect packet fields or send a template generated packet of a constant length.

b. IP Do Not Fragment

IP Do not fragment is a bit set in a TCP/IP packet to ensure that a packet will not be broken up by network systems as it travels to its destination. The implementation of this protocol function can vary, and in some cases this can cause errors in connections between some systems.

c. TCP Initial Window Size

This option is used to advertise that buffer window that is available for TCP packets on the host. This allows for maximum throughput of network packets and minimizes delay when large amounts of data are being sent. Not all OSes correctly implement this option to reflect the sizes of their buffers. Some OSes also simply echo received size in return messages, others use a constant window size, and some OSes have limited memory and must use a much smaller window size.

3. Predictable Value

In some cases the OS will choose predictable values for fields that are convenient for the OS and not specified in the protocol. These predictable values are found in the advertised window size. The OS uses a memory management system to allocate space to hold TCP/IP packets as they are received. By using a standard allocation method the OS can efficiently use its memory and maximize the potential throughput of the packets, but this also means that the advertised window size can reveal details of the type and version of the OS.

Because TCP/IP packets are transported over an unreliable network, each TCP/IP packet also has sequence numbers to help the packets be reconstructed in the right order and to rebuild packets from fragments that may have occurred. These sequence numbers are increased with each packet that is sent or received. Since each connection needs a new set of sequence numbers the OS needs to generate a starting or seed number for each connection. Each OS creates these sequence numbers in a different way. Some methods of generating sequence numbers creates the potential for session hijacking by an attacker guessing the sequence number and submitting malicious TCP/IP packets with forged headers and sequence numbers to take over the connection of another user. Some OSes use sequence numbers that have only 65,535 unique values other use up to

10,737,254,420 (65,535 * Maximum Window Scale of 16,384) possible values. The larger the number of values the safer the system is from session hijacking.

D. OS FINGERPRINTS USES

An OS Fingerprint can contain a unique description of the OS and version running on the host. These variations usually have little to no effect on the operation of the system itself. They tend to only affect the way the system is optimized to operate over the network. But different versions and types of OS will still have variations that result in changes visible in the TCP/IP packets that would normally be unnoticed to the users of a system.

Because an OS Fingerprint can be specific as to which version an OS is, it can be used to identify specific systems that contain known vulnerabilities. The OS Fingerprint can be more accurate than simply “Banner Grabbing”, since banners can easily be forged or even removed entirely. In addition, OS Fingerprinting works on systems that don’t even have services that normally provide banners to grab.

There are currently many network tools that capture and use OS Fingerprints. These tools are used by hackers and system administrators for different purposes. An attacker would look for known weaknesses by OS and version, where a system administrator may use OS Fingerprints to enhance his firewall settings and restrict OS types that are known to be vulnerable to worms from using certain service (i.e. Email). Either way OS Fingerprints are becoming a useful artifact of a network connection that users may or may not want to be available or captured by other systems.

1. Nmap

Nmap was one of the first network port scanning tools to use OS Fingerprinting from TCP/IP packets [13]. As a port scanner, Nmap is designed to search a target host for all open ports. This allows the user to determine what services may be available on a system. It also allows an administrator to determine which services have correctly been locked down and which have not. Nmap was designed as an attacker’s tool by including specialized scan options such as spoofing false sources for scanning packets, scanning with half open connections, scanning with FIN packets and by scanning slowly to avoid detection. By adding OS Fingerprinting Nmap became the de facto standard tool for network scanning.

Nmap was first released September 1997 and has archived versions from 1.25 to 2.53. Over the past several years it has grown and changed until now. The current version 3.8-1 has added new capabilities and functionality over the original.

Nmap uses a series of TCP/IP and UDP packet messages to collect the OS Fingerprint from a remote host. The responses are compared to an extensive library of over 1300 known OS Fingerprints that have been collected by users across the internet. Recently Nmap has built-in additional capabilities such as port application service detection by attempting to grab the connection banners and comparing that to a library of known banners as well.

2. p0f

p0f is a similar OS Fingerprint detection tool [14] , but differs from Nmap in that it uses passive detection. That means instead of sending out collection requests to the target machine, it listens to messages coming to the local machine from remote machines and at that point attempts to detect the OS Fingerprint from captured TCP/IP packets[15]. This approach protects the attacker from having his scan detected. Some firewalls detect network scans and block hosts that try to run active network scans.

II. OS FINGERPRINT CHANGING

A. HOW OS FINGERPRINTS CAN BE CHANGED

Because TCP/IP fingerprinting effectively broadcasts your OS identification to any network host that can send or receive packets, it becomes important to be able to control access to or limit accuracy of the TCP/IP fingerprint of a system from other untrusted systems and users.

A TCP/IP fingerprint can be partially protected by using a firewall to block certain types of packets [7, 16]. Sometimes a firewall will effectively block a TCP/IP fingerprint, but not always. Nmap can use partial matches and account for packets that may be dropped by a firewall. A complete TCP/IP fingerprint can be made from only one open port to the host machine even through a NAT (network address translation) forwarded port [17].

Because TCP/IP fingerprinting uses a library of known OS Fingerprints small changes in the TCP/IP operations of the host computer can conceal the OS by making only minor changes to its TCP/IP operation. Any change outside the bounds of known OS Fingerprints will result in an unidentified OS Fingerprint.

Although this may defeat some attackers from quickly determining the correct OS Fingerprint for the modified system, a more persistent attacker will be able to compare the fingerprint collected with known fingerprints to determine which OS is the "most likely" match. In addition, some simple changes to the OS are more common to be used by other systems and therefore they may already be in the fingerprint library used. Many OSes have configuration options that can change the OS Fingerprint and may be useful for certain network configurations.

So attempting to change the OS Fingerprint of a system is best achieved by closely matching a known system rather than making changes to just change the original OS Fingerprint. Changing the system away from the true system will only lead to OS Fingerprinting tools that will use statistical estimation to determine the most likely system based on the type of changes found.

B. DYNAMIC CHANGES

A network fingerprint does not have to be a constant and unchanging value. The methods used to determine an OS Fingerprint simply use certain types of TCP/IP packets to gather known responses and then compare those responses to a lookup table of known values for known OSes. It is possible for an OS to change the way it responds to these TCP/IP packets dynamically during operation.

It's important to note that many operating systems have customizable settings for how the network stack operates. These settings can allow various aspects of the TCP/IP fingerprint to be changed. For instance it may be possible to configure the amount of memory that the system uses for each network connection. By changing the memory allocation model, you may increase or decrease performance, but as a side effect, the Window Size and Window Scale values of the OS Fingerprint may change as well. Other system settings allow administrators to configure other details of the OS finger print. An example of this is in Linux system it's possible to turn off the TCP option for Time Stamp or the Window Scaling by using a "sysctl" command.

Many operating systems have configurations and settings that affect the way TCP/IP packets are created and sent out on the network. Many of these settings and configurations are available to system administrators to change dynamically. But not all aspects of the TCP/IP fingerprint can be changed simply by modifying these settings. Some fingerprint characteristics can only be changed by directly modifying the original system kernel or by making changes to the TCP/IP packets in an application that run like a firewall on the system. Changing the OS kernel will require the system to be rebooted to make the changes effective. Changing the OS setting does not always require the system to be rebooted. Applications like firewalls are designed to allow changes to be made dynamically with out rebooting.

C. RELATED WORKS

There are several different ways to control or protect the OS Fingerprint of a system. All of the approaches focus on changing the TCP/IP packets that are transferred on the network.

1. Packet Creation

This is the approach used in this effort. This approach focused on modifying the TCP/IP packets as they are created in the OS kernel prior to being sent out over the network. This approach can work on an OS that has full access to configuration and source code or the OS is designed with application interfaces to modify packets prior to sending them out. Because OS Fingerprinting uses both TCP/IP packet and UDP packet, multiple changes will be needed to fully control the OS Fingerprint.

2. Packet Filtering

This approach uses firewall rules and interfaces to capture packets as they are leaving a system whether they are created locally or just passing through a gateway. TCP/IP packets can either be modified or blocked by a filter.

When modifying packets, care must be taken to not break operational requirements of the protocol or system. Response packets must also be captured and rewritten to match the original unmodified packets so the original system will recognize them as correct responses. Blocking packets can also confuse OS Fingerprint scans by not allowing them to capture the complete OS Fingerprint and may result in an ambiguous OS Fingerprint result [12, 18].

One example of packet filtering for OS Fingerprint protection is the OpenBSD PF firewall configuration. PF firewall can correct or block TCP/IP packets that fail to correctly implement the RFC and other restrictions [16]. This firewall can reassemble TCP/IP fragments before they are processed by the OS to avoid possible errors these fragments may cause and also allow filters to block or correct other TCP/IP packets with fields in invalid states. The firewall can block invalid packets such as a SYN/FIN, ACK/FIN and TCP/IP options in packets other than SYN. The firewall grabs the TCP/IP packet and checks it to be sure its valid and normalizes it before allowing it to be processed by the rest of the system. This can limit the ability of any OS detection tool to gather an OS Fingerprint of a system behind the firewall.

3. Packet Correction and Verification

This approach works on a similar approach as packet filtering, but instead of arbitrarily modifying or blocking packets, it focuses on correcting errors and invalid values in packets. A packet correction and verification system will block invalid RST packets, reassemble fragmented packets, and can strip out invalid TCP options in packets where they are not allowed [10].

III. PROTOTYPE AND BEYOND

A. LINUX TO WINDOWS

To demonstrate the principles of modification of an OS Fingerprint modification, a prototype system called “Neaconing” was developed. “Neaconing” is a term created after the radar term “Meaconing” which means to broadcast false beacon signals to confuse and mislead navigation systems. “Neaconing” is an attempt to falsify a network fingerprint of an OS to confuse and mislead network scanning systems. The prototype system was created by modifying an existing operating system so that it matched the OS Fingerprint of another. There were many choices for operating systems to use in this prototype. Choosing the right system can make the process of modifying the OS fingerprint harder or easier.

1. Selecting the OS Fingerprint to Modify

When selecting an OS for modification as a prototype for OS Fingerprint modification, it is important that the OS be easily modified, that the details of the implementation or source code be inspectable, and the software and hardware for the OS needs to be available. For this project the Linux OS was selected because it met these requirements well.

The Linux OS is greatly available to modification because it was created and is maintained by various contributors around the world as an open source project. Linux demonstrates its ability to be modified by the user with its ability to be maintained by a diverse group of developers working collectively on the system independently world wide.

Linux is an open source system, which means the entire OS source code was available to change and recompile as a custom system. It may be possible to apply these same principles to other OSes but to do so would require purchasing the software license and gaining access to the source code when changes would be required.

Linux OS is easily acquired and the hardware, software and licensing requirements of a system to use, modify, or compile a custom modified Linux OS kernel are easily available. The Linux OS supports many different architectures of hardware and includes the i386 architecture which most home PCs use as well as PPC, Alpha and others. Additionally, being open source also makes the Linux OS inspectable. The entire source code of the Linux kernel can be downloaded by anyone free of charge. There are other OSes that are also open source (i.e. NetBSD, FreeBSD, OpenBSD), and while these may have also made good candidates for the prototype system, many of the other OSes have made design decisions (such as security in OpenBSD and compatibility in NetBSD) that may have made some modifications more complicated than in a Linux OS. It would also be possible to modify a proprietary OS, but in order to do so special licensing, and nondisclosure agreements would need to be arranged. In addition the results might not even be publishable.

2. Selecting the OS Fingerprint to Match

The target OS was chosen to be a Windows NT. Windows NT has many advantages as a target machine. The primary advantage of using a Windows system as a target OS Fingerprint is that currently Windows is the dominate OS on desktop.

By choosing an extremely obscure target OS, an attacker would have reason to doubt the results of the scan. Who would believe that a network machine serving as a service provider would be running on a "Sega Dreamcast game console"?

Windows has evolved over many versions. The Windows NT OS was a common OS in 1999 when the effort was begun, but the principles used to modify the OS Fingerprint would apply equally well to other versions of Windows such as Windows 2000, XP, and 2003. Additionally, OS Fingerprints tend to change over time. Each version of Nmap releases updates to the OS Fingerprint database. As new OS configurations or an OS is installed on additional hardware, some times the OS Fingerprint can change slightly as memory or CPU speed change.

3. Collecting and Describing the Original OS Fingerprint

The first step in changing a Linux System's OS Fingerprint is to determine what the exact fingerprint is. The fingerprint of the initial system can be collected by using the Nmap tool. On a Linux machine, Nmap can be installed by downloading the source code and compiling it, or by downloading an RMP (Red Hat Packet Manger) file and installing it with the command "rpm –Uvh". Many distributions already include Nmap as a system tool, or as an installation optional packet.

In order to collect an OS Fingerprint in Nmap the Nmap command must be run from a root or admin level console. This is necessary because Nmap needs to have access to create arbitrary TCP/IP packets directly in the OS kernel. Some of the test packets Nmap sends out are invalid and could not be created though non-privileged system calls.

In the original test system was a SuSE 9.2 Linux default installation desktop computer running Linux 2.6.X kernel. The results of the Nmap scan report the following:

Table 1. Nmap results for test system OS.

```
#nmap –O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-03-12 22:46 UTC
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
Uptime 0.023 days (since Sat Mar 12 22:14:23 2005)

Nmap run completed -- 1 IP address (1 host up) scanned in 6.472 seconds
```

Occasionally Nmap may return different results for the same OS scan. This is because it may not have enough scan results to distinctly identify the exact OS from another. The Nmap scan returns this alternative result on more than one occasion:

Table 2. Nmap additional results for test system OS.

```
#nmap -O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-03-12 23:01 UTC
Insufficient responses for TCP sequencing (3), OS detection may be less accurate
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.0 - 2.5.20, Gentoo 1.2 linux (Kernel 2.4.19-gentoo-rc5), Linux 2.4.20, Linux
2.4.20 - 2.4.22 w/grsecurity.org patch, Linux 2.5.25 - 2.6.3or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
Uptime 0.033 days (since Sat Mar 12 22:14:23 2005)

Nmap run completed -- 1 IP address (1 host up) scanned in 2.455 seconds
```

Nmap determined the OS of this system by looking up the results of its OS scan in its database. The Nmap OS Fingerprint database is located at:

/usr/share/nmap/nmap-os-fingerprints

By using the responses from the Nmap OS scan we can create a list of the possible matches for our OS in the Nmap fingerprint database. There are over 1300 distinct OS Fingerprints in the Nmap fingerprint database. There are 277 different Linux OS Fingerprints and up to five of these OS Fingerprints matched the test system.

Table 3. Nmap OS Fingerprint database entries.

```
Fingerprint Linux 2.4.0 - 2.5.20
Class Linux | Linux | 2.4.X | general purpose
Class Linux | Linux | 2.5.X | general purpose
TSeq(Class=RI%gcd=<8%SI=<2D870AA&>10000%IPID=Z|C|I|RD%TS=100HZ|U)
T1(DF=Y%W=5B4|F98|1140|11AC|12CC|16A0|1680|2D24|4000|474C|7E18|7EA0|7FFF%ACK=S++%Flags=AS%Ops=MNNTNW|MNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=5B4|F98|1140|11AC|12CC|16A0|1680|2D24|4000|474C|7E18|7EA0|7FFF%ACK=S++%Flags=AS%Ops=MNNTNW|MNNT)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=Y|N%TOS=0|8|14|20|28|38|40|C0|C8%IPLEN=164%RIPTL=148%RID=E%RIPCK=E|F%UCK=E|F%ULEN=134%DAT=E)

# I don't put “Gentoo” as the vendor for the classification because it
# makes output ugly when there are a bunch of Linux matches due to
# firewalling or similar problems.
Fingerprint Gentoo 1.2 linux (Kernel 2.4.19-gentoo-rc5)
Class Linux | Linux | 2.4.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<1BF1FC0&>4788F%IPID=RD%TS=1000HZ)
T1(DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Fingerprint Linux 2.4.20
Class Linux | Linux | 2.4.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<18D4252&>3F8B9%IPID=Z%TS=100HZ)
T1(DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=16A0%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=8%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

# Linux 2.4.20-gentoo-r5 w/grsecurity
Fingerprint Linux 2.4.20 - 2.4.22 w/grsecurity.org patch
Class Linux | Linux | 2.4.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=RD%TS=100HZ)
T1(DF=Y%W=5B4|16A0|7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=5B4|16A0|7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0|C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)

# Linux matrix 2.6.3-gentoo-r2 x86
Fingerprint Linux 2.5.25 - 2.6.3 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)
Class Linux | Linux | 2.4.X | general purpose
Class Linux | Linux | 2.5.X | general purpose
Class Linux | Linux | 2.6.X | general purpose
TSeq(Class=RI%gcd=<6%SI=<244F6FE&>5CF30%IPID=Z%TS=1000HZ)
T1(DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

By looking up the “Linux 2.5.25 - 2.6.3” or the “Gentoo 1.2 Linux 2.4.19 rc1-rc7” we can see the most common match Nmap found for the local scan. Some of the other fingerprints also match and these are also listed as possibilities. This allows an attacker to identify the most likely OS from a list of all. But each OS in our scan reported Linux as the base OS and the only variation was for which distribution or configuration it used.

Each of these five fingerprints actually represents multiple possible fingerprints. Each line represents a test or packet response. Within these packet responses there may be multiple possible values for each field. These are separated by the symbol “|” for OR, and other expressions like “<“ less than, and “>“ greater than.

When Nmap is unable to identify the OS of a scanned system, it will report the fingerprint to the console so it can be collected and added to the Nmap OS database once it is known. In order to collect the exact OS Fingerprint of the test system, the Nmap OS Fingerprint database needs to be cleared of all valid values. This can be done by erasing all the data in the /usr/share/nmap/nmap-os-fingerprint file.

Table 4. Clear Nmap OS Fingerprint database.

```
neuron: #echo ““ >/usr/share/nmap/nmap-os-fingerprint
```

Once the database has been cleared, Nmap will report the exact fingerprint it detects on any system.

```
#nmap –O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-03-12 23:12 UTC
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
No OS matches for host (If you know what OS is running on it, see http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.70%P=i586-suse-linux%D=3/12%Time=4233775F%O=22%C=1)
TSeq(Class=RI%gcd=1%SI=3041E7%IPID=Z%TS=1000HZ)
TSeq(Class=RI%gcd=1%SI=3041D1%IPID=Z%TS=1000HZ)
TSeq(Class=RI%gcd=1%SI=3041DA%IPID=Z%TS=1000HZ)
T1(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

Uptime 0.040 days (since Sat Mar 12 22:14:23 2005)

Nmap run completed -- 1 IP address (1 host up) scanned in 9.951 seconds
```

Table 5. Nmap actual OS Fingerprint of original test system.

4. Collecting and Describing the Target OS Fingerprint

There are 96 different fingerprints for various builds of windows from 3.1 to Longhorn, including PDAs and Pocket PC versions.

There are 277 different fingerprints for various builds of Linux from embedded devices to printers and kernel 1.X to kernel 2.6.X.

By comparing our actual OS Fingerprint we can determine which of the target OS Fingerprints are easiest to match.

Many of the Windows systems have required responses to the T2 test which the test system does not respond to as currently configured. These fingerprints are not good candidates.

The fingerprint that was most similar to the test OS Fingerprint was the “Windows NT 3.51 SP5” fingerprint.

Table 6. Nmap database fingerprint of target OS Fingerprint.

```
# Windows NT 3.51 SP5
Fingerprint Microsoft Windows NT 3.51 SP5, NT 4.0 or 95/98/98SE
Class Microsoft | Windows | 95/98/ME | general purpose
Class Microsoft | Windows | NT/2K/XP | general purpose
TSeq(Class=TD|RI%gcd=1|2|3|4|5|A|14|1E|28|5A%SI=<1F4%IPID=BI|RPI|RD%TS=U|0)
T1(DF=Y|N%W=2017|3908|16D0|860|4470|61A8|7FFF|8000|869F|9C40|FAF0%ACK=S++%Flags=A|AS%Ops=|M|MNWNNT)
T2(DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y|N%W=2017|3908|16D0|860|4470|61A8|7FFF|8000|869F|9C40|FAF0%ACK=S++%Flags=AS%Ops=M|MNWNNT)
T4(DF=N%W=0%ACK=S++|O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=S++|O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++|S%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)
```

5. Identifying the Necessary Changes to Match OS Fingerprints

The first step to modify our OS Fingerprint is to identify the differences between our OS Fingerprint and the target OS Fingerprint. The target OS has many possible combinations of values that are valid. For instance the "W" or window size of the T1 and T3 responses can be 2017 or 3908 or 16D0 or 860 or 4470 or 61ab or 7fff or 8000 or 869f or 9c40 or faf0. By selecting this fingerprint as the target these "W" values already match the initial OS Fingerprint value of 7fff.

In test T1 and T3 there are obvious differences. The "Ops" or TCP Options portion of the fingerprint have different values. The test OS values are "MNNTNW" and the target values are "MNWNNT". Nmap not only looks for which values the fingerprint has (such as M, W, N and T), but it also looks at the order. So even though we have all the same options the order is different.

Table 7. Test system FP, Target system FP and actual changes

```
test      TSeq(Class=RI%gcd=1%SI=3041E7%IPID=Z%TS=1000HZ)
targ      TSeq(Class=TD|RI%gcd=1|2|3|4|5|A|14|1E|28|5A%SI=<1F4%IPID=BI|RPI|RD%TS=U|0)
act       TSeq(Class=RI%gcd=1%SI=<1F4%IPID=BI|RPI|RD%TS=U|0)

test      T1(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
targ      T1(DF=Y|N%W=2017|3908|16D0|860|4470|61A8|7FFF|8000|869F|9C40|FAF0%ACK=S++%Flags=A|AS%Ops=|M|MNWNNT)
act       T1(DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)

test      T2(Resp=N)
targ      T2(DF=N%W=0%ACK=S%Flags=AR%Ops=)
act       T2(Resp=N)

teset     T3(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=MNNTNW)
targ      T3(Resp=Y%DF=Y|N%W=2017|3908|16D0|860|4470|61A8|7FFF|8000|869F|9C40|FAF0%ACK=S++%Flags=AS%Ops=M|MNWNNT)
act       T3(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)

test      T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
targ      T4(DF=N%W=0%ACK=S++|O%Flags=R%Ops=)
act       T4(DF=N%W=0%ACK=O%Flags=R%Ops=)

test      T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
targ      T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
act       T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

test      T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
targ      T6(DF=N%W=0%ACK=S++|O%Flags=R%Ops=)
```

```
act           T6(DF=N%W=0%ACK=O%Flags=R%Ops=)

test          T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
targ          T7(DF=N%W=0%ACK=S++|S%Flags=AR%Ops=)
act           T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)

test          PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
targ          PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E|F%ULEN=134%DAT=E)
act    PU(Resp=N)
```

Each of these lines of the Nmap fingerprint is described in detail on the Nmap Web site http://insecure.org. But a brief description of each fingerprint component is given here to assist the reader.

Table 8. Fingerprint Components.

TSeq – This component of the fingerprint reports the TCP/IP sequencing number generation methods used.

T1 – This component is a SYN packet to an open TCP port.

T2 – This component is a NULL packet to an open port.

T3 – This component is a SYN/FIN/URG/PSH packet to an open port.

T4 – This component is an ACK packet to an open port.

T5 – This component is a SYN packet to a closed port.

T6 – This is an ACK packet to a closed port.

T7 – This is a FIN/URG/PSH packet to a closed port.

PU – This is a port unreachable response packet.

Each of these packets has a complete list of options included to solicit a response from the target system of which options it supports. Other TCP/IP fields are also used such as TCP/IP flags, advertised window size, acknowledge sequence number and Do Not Fragment bit.

6. Process for Changing the OS Fingerprint

It would be possible to rewrite the code in the kernel that generates the options so they list in a different order, but because the target fingerprint also accepts the “M” only option, we can also turnoff the “T” timestamp and “W” window scaling options and create a match.

The “T” and “W” options can be configured dynamically in Linux either by using the “sysctl” command or by editing the “/proc” file associated with the system variable.

The “sysctl” command to change these two values is

Table 9. OS Modification 1 - Linux system Sysctl modification commands

```
neuron:  # sysctl net.ipv4.tcp_timestamps=0
net.ipv4.tcp_timestamps = 0
neuron: # sysctl net.ipv4.tcp_window_scaling=0
net.ipv4.tcp_window_scaling = 0
```

The other way these variables can be changed is to write a “0” for off to the “/proc” files

/proc/sys/net/ipv4/tcp_timestamps

/proc/sys/net/ipv4/tcp_window_scaling

Table 10. OS Modification 1 - Linux system file modification commands

```
neuron: # echo 0 > /proc/sys/net/ipv4/tcp_timestamps
neuron: # echo 0 > /proc/sys/net/ipv4/tcp_window_scaling
```

Once these sysctl changes have been made, Nmap reports a different fingerprint. This is still a Linux fingerprint, so someone has already realized how this small change can defeat Nmap.

Table 11. Nmap results for test OS modification 1

```
neuron:~ # nmap -O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-03-26 19:10 UTC
Insufficient responses for TCP sequencing (3), OS detection may be less accurate
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
Device type: general purpose
Running: Linux 2.4.X
OS details: Linux 2.4.23-grsec w/o timestamps, Linux 2.4.7 (x86)

Nmap run completed -- 1 IP address (1 host up) scanned in 2.662 seconds
```

Running Nmap with an empty database shows the complete fingerprint with the changes made to the TCP Options.

Table 12. Nmap actual fingerprint for test system OS modification 1

```
neuron:~ # nmap -O localhost

No OS matches for host (If you know what OS is running on it, see http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.70%P=i586-suse-linux%D=3/12%Time=4233823B%O=22%C=1)
TSeq(Class=RI%gcd=1%SI=41DE87%IPID=Z%TS=U)
TSeq(Class=RI%gcd=3%SI=15F4D6%IPID=Z%TS=U)
TSeq(Class=RI%gcd=1%SI=41DE9C%IPID=Z%TS=U)
T1(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=Y%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=Y%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

At this point many of the T responses are nearly matching the target OS Fingerprint. Looking at each fingerprint result, the differences are as follows.

Table 13. Fingerprint with Changes.

T1 is already a match since DF=N matches and the windows size of 7FFF is compatible.

T2 is already a match. Response is optional and we currently do not respond.

T3 is a match since DF=N matches and the windows size of 7FFF is compatible.

T4 still requires a change to DF=N, but the ACK can be 0 or S++.

T5 still requires a change to DF=N.

T6 still requires a change to DF=N, but the ACK can be 0 or S++.

T7 still requires a change to DF=N.

So the primary change left is to change the DF=Y setting in the test Linux OS Fingerprint to match the DF=N in the Windows target OS Fingerprint.

This change required modifying the Linux kernel. The Linux kernel is open source and free to download and modify. For most Linux distributions the kernel source code can be installed during the initial installation or it can be installed from an RPM package. The source code can also be downloaded from various Web and ftp sites. There are many different versions of the Linux kernel. This paper discusses changes made to the Linux kernel version 2.6.8-24.10.i568.

Before you can modify the source code and build a new kernel, all the dependences needed to build the kernel from source must also be installed on the system. These include the gcc compiler and the various kernel-devel rpms depending on the configuration of the system.

Once the kernel source is installed it will be located in the /urs/src/ directory under the linux folder. This folder is actually a link to the actual kernel folder in the same directory (/usr/src/). The actual kernel folder contains the version number in the folder name.

Before making changes to the kernel, the default kernel should be compiled and installed to verify that it works correctly. The process for compiling a kernel for Linux can be quite tricky. There are many hundreds of options to set and choices to be made. There are options to build components into the kernel or have them load as modules.

It’s also possible to rebuild the kernel using the “old” configuration options once the configuration has been initially set.

Table 14. Linux kernel rebuild from source code

```
neuron:/usr/src/linux #make mrproper
neuron:/usr/src/linux #make oldconfig
neuron:/usr/src/linux #make dep
neuron:/usr/src/linux #make clean
neuron:/usr/src/linux #make bzImage
neuron:/usr/src/linux #make modules
neuron:/usr/src/linux #make modules_install
```

The output of this kernel building is extensive and included as an attachment. The final product of the kernel build is a 1.5MByte kernel file “bzImage”. On a development system running on a 1.4Ghz processor the kernel build time was approximately 54 minutes.

This final “bzImage” file needs to be installed into the /boot directory of the system along with its System.map file.

In this case temporary KERNEL_VERSION names of “aaron1” and “aaron2” were used to differentiate test kernels from default kernels.

Table 15. Linux kernel install

```
cp arch/i386/boot/bzImage /boot/bzImager-aaron2
cp System.map /boot/System.map-aaron2
ln -s /boot/System.map-aaron2 /boot/System.map
```

The kernel is the foundation element of the OS and is loaded directly from the boot loader. In order for the boot loader to know where to find the kernel the boot loader must be configured to use the new kernel. In a SuSE 9.2 system there are two possible boot loaders to use by default. This test used the Grub boot loader. Each time a new kernel is installed it must be added to the Grub boot loader by inserting these three lines to the Grub menu file /boot/grub/menu.lst.

Table 16. Linux bootloader configuration

```
title LINUX-aaron2
kernel (hd0,2)/boot/vmlinuz-aaron2 root=/dev/hda3 selinux=0 resume=/dev/hda2 showopts
splash=silent desktop elevator=as
initrd (hd0,2)/boot/initrd
```

Once the new Linux kernel is built and installed and configured in the boot loader, the system must be restarted. There is no other way to remove an old kernel and start a new one. During boot up the new kernel is selected from the possible kernels and the system start up and runs without error.

Now it's time to modify the kernel to change the OS Fingerprint. The first task is to disable IP Do Not Fragment. There are many places where the Do Not Fragment bit is used and set in the kernel source code, but after careful study and experimentation the declaration of the Do Not Fragment in the /usr/src/linux/include/net/ip.h file was found to be the root configuration and instantiation location.

By modifying the ip.h file so that the ip_do not_fragment method will always return false, all TCP/IP packets will report DF=N. Below is the code section in the /usr/src/linux/include/net/ip.h file that needs to be changed to do this.

Table 17. OS modification 2 - IP Do Not Fragment code change

```
//ip.h near line 185

static inline
int ip_dont_fragment(struct sock *sk, struct dst_entry *dst)
{
return 0;
        //return (inet_sk(sk)->pmtudisc == IP_PMTUDISC_DO ||
        //      (inet_sk(sk)->pmtudisc == IP_PMTUDISC_WANT &&
        //       !(dst_metric(dst, RTAX_LOCK)&(1<<RTAX_MTU))));
}
```

Once this change is made, the kernel and modules must be rebuilt. Then when the new kernel is installed and configured in the boot loader, the system must be rebooted so the new kernel can be loaded. Once the new kernel is loaded, the Nmap scan reports the following:

Table 18. Nmap actual fingerprint for OS modification 2

```
neuron: #nmap –O localhost

No OS matches for host (If you know what OS is running on it, see http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.70%P=i586-suse-linux%D=3/13%Time=4233D7FB%O=22%C=1)
TSeq(Class=RI%gcd=1%SI=4E7C23%IPID=I%TS=U)
TSeq(Class=RI%gcd=1%SI=4E7C56%IPID=I%TS=U)
TSeq(Class=RI%gcd=1%SI=4E7C3E%IPID=I%TS=U)
T1(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=C0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

No known fingerprints match system fingerprint. At this point our modifications have defeated Nmap’s ability to correctly identify the test system. Although the changes are small enough, a reasonable attempt to guess the OS would still lead us to conclude it was a Linux OS. This new fingerprint shows many actual matches for all the T responses of the target system fingerprint.

Table 19. Remaining Fingerprint Issues

T1= match possible.

T2= match by absence.

T3= match possible.

t4= match possible.

t5= match possible.

t6= match possible.

t7= match possible.

This leaves the first TSeq and the last PU responses to be modified. The TSeq is primarily a test for the sequence number of the TCP/IP packets. Few scans actually gather enough details to detect how these sequence numbers are generated. It will not be necessary to change the TSeq for the OS Fingerprint to match since both OSes are relatively similar.

Table 20. TSeq Fingerprint Differences

```
SI  =4E7C3E      need <1F4
IPID=I           need BI | RP | RD   (all are I or Increasing types)
```

The PU response is a port unreachable response message over the UDP protocol. The difference between the target OS and the test kernel response is the IPLEN or IP packet length.

Table 21. PU Fingerprint Differences

```
PU(DF=N%TOS=0%IPLEN=38 %RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
PU(DF=N%TOS=0%IPLEN=164%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
IPLEN=164     need 38
```

Because most OSes use a template for sending this standard ICMP response, these lengths are different. There are several different ways to change this value, but all require a rewrite of the UDP code in the kernel. The target OS also can be detected with no response to this test message. This change requires the kernel be further modified so that it will silently ignore all UDP messages that would normally generate a Port Unreachable message.

After careful inspection of the kernel source code many file used UDP messages and IP packet length. The file /usr/src/linux/included/net/ipv4/udp.c has a method udp_rcv() that routes UDP packets. If the UDP packet does not have a valid port to be delivered to it falls out the bottom of the method and is dropped. But just before the end of the method there is a call to icmp_send(skb, ICMP_DEST_UNREACH, ICMP_PORT_UNREACH, 0). This generates and sends the port unreachable message in reply to these soon to be dropped packets. By commenting out this line, all port unreachable messages will be silently dropped with no reply.

Table 22. OS modification 3 - UDP Port Unreachable code change

```
///usr/src/included/net/ipv4/udp.c about line 1172
            //icmp_send(skb,ICMP_DEST_UNREACH, ICMP_PORT_UNREACH, );
```

This change requires that a new kernel be built. This kernel was called version "aaron2" and installed in the Grub loader.

7. Resulting OS Fingerprint

With this second kernel rebuild running Nmap OS detection returns the following

Table 23. Nmap results for test OS modification 3

```
neuron: # nmap –O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2005-03-26 14:57 UTC
Insufficient responses for TCP sequencing (1), OS detection may be less accurate
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows NT 3.51 SP5, NT 4.0 or 95/98/98SE

Nmap run completed -- 1 IP address (1 host up) scanned in 2.396 seconds
```

To an Nmap scan, this Linux 2.6.X system appears to be a windows NT 3.51 SP 5 system.

Table 24. Nmap actual fingerprint for test OS modification 3

```
neuron:~ # nmap -O localhost

Starting nmap 3.70 ( http://www.insecure.org/nmap/ )
at 2005-03-27 20:49 UTC
Insufficient responses for TCP sequencing (1), OS
```

```
detection may be less accurate
Insufficient responses for TCP sequencing (1), OS
detection may be less accurate
Insufficient responses for TCP sequencing (2), OS
detection may be less accurate
Interesting ports on localhost (127.0.0.1):
(The 1656 ports scanned but not shown below are in
state: closed)
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind
631/tcp open  ipp
No OS matches for host (If you know what OS is running
on it, see
http://www.insecure.org/cgi-bin/nmap-submit.cgi).
TCP/IP fingerprint:
SInfo(V=3.70%P=i586-suse-linux%D=3/27%Time=42471C73%O=22%C=1)
T1(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T2(Resp=N)
T3(Resp=Y%DF=N%W=7FFF%ACK=S++%Flags=AS%Ops=M)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=N)


Nmap run completed -- 1 IP address (1 host up) scanned
in 9.982 seconds
```

This new fingerprint falls within the known possible fingerprints of a Windows NT system. Whether or not this is a common fingerprint for a Windows NT system, or even an actual fingerprint and not a mismatch of partial fingerprints remains to be seen. Without having all possible Windows NT systems to generate all possible Windows NT fingerprints, the Nmap fingerprint database will have to make the final determination. The Windows NT OS Fingerprint has changed from Nmap version 2.54 to version 2.73. The changes made in this effort do not work for all versions of Nmap, and may not work for future versions. The only way to guarantee a match is to collect an OS Fingerprint of a target system and exactly match it and not depend on Nmap fingerprint database values.

B. COMPUTER LEVEL PROTECTION

It’s possible for each computer to be modified to change its OS Fingerprint. This can be done by following some of the guidance above such as “sysctils” or actually modifying the system kernel.

While most types of operating systems have some capability similar to “sysctils” not all allow users to access the source code of the kernel to modify and build specialty kernels that can disguise the OS Fingerprint. Specifically proprietary OSes like Windows and HP Unix do not allow developers access to modify the OS kernel with out special arrangements.

C. GATEWAY PACKET SCRUBBING

Another approach other than modifying each system individually to disguise the OS Fingerprint is to have the gateway or firewall computer intercept and modify the packets as they are passed through to remove or modify the parts of the TCP/IP packet that would reveal the OS Fingerprint. In part a normal firewall can do some of this since it may simply block much of the TCP/IP packets that a network fingerprint scan would need to create. This is because many of the packets used to scan for the OS Fingerprint actually contain invalid sequences such as ACK/FIN and options in packets other than the SYN, or UDP packets to closed ports. While a firewall may be a good start, it clearly isn’t enough since many OS Fingerprints can be determined with only one open port to the host. The solution is to have the gateway computer process each packet and filter out specific values that are optional and modify packet header values that are changeable.

D. FINGERPRINTING IN OTHER SYSTEMS

1. Hubs & Routers

While most users are blissfully unaware of how network traffic is transferred from host to host, there is a complicated world that consists of various network devices such as Firewalls, Hubs, Routers, and Switches that pass network packets around and track paths between points. Each of these devices is actually a computer system similar to a desktop host and many internet attackers look to these as a first line of an attack. A computer network can be protected by locating it behind a firewall. The firewall will limit network traffic into and out of the LAN. To an attacker, this firewall is a gateway into the network. By finding a vulnerability in the firewall and exploiting it, the attacker can gain access to the entire network and all the machines protected by it. Since these devices act on the TCP/IP protocol layer they too can be scanned and they too can be identified by Nmap and other tools by their OS Fingerprint. A quick look at some of the 1300 OS Fingerprints reveal hundreds of firewalls, routers and switches. These systems need to have their OS Fingerprint protected as much, if not more, than the systems they protect.

2. Modems

Modems are connection points for computers into a network. Traditionally modems have been used for connecting a computer to another computer over traditional phone lines. Recently with the increase in broadband connection through DSL and cable systems, other types of modems have come to the market. These modems connect computers over various media into the larger internet. DSL modems use special high bandwidth phone lines and cable modems use the coaxial cable lines that also carry TV signals. While these systems connect through proprietary networks, a quick search of the Nmap fingerprint database reveals over twenty different cable/DSL modem fingerprints. These devices are visible from the TCP/IP network they provide connection to and from. Additionally, these devices can provide services such as DHCP, firewall and port forwarding. Many modems provide configuration through a built in Web service. These network services and the modems’ network locations result in each device having an OS Fingerprint that is detectable and useful to a potential attacker. A quick search in the Nmap fingerprint library shows that these devices are known and tracked. To a network attacker the modem can be a gateway into a larger network, and therefore modems too should protect their TCP/IP OS Fingerprint.

IV. CONCLUSION

A. SUMMARY

OS Fingerprints are a useful tool for a network attacker to identify a machine that may be vulnerable to a specific attack. Network fingerprints exist because of default configuration and operating system variations. Because OS Fingerprints can be detected and modified, it is possible to change the OS Fingerprint of one machine to look like another. The prototype developed shows one possible way to modify a Linux 2.6.X system to have the same OS Fingerprint as a Windows NT system. There are multiple methods to change the OS Fingerprint. In the prototype developed in this effort, system configurations were changed and a custom modified OS kernel was created. It is also possible to change the OS Fingerprint by modifying the TCP/IP packets at a router, switch or gateway machine. Some systems already provide some of these capabilities such as Scrub from OpenBSD. These tools modify the TCP/IP packet and can block packets that are used for OS Fingerprinting since some of these packets are nonfunctional twists of the TCP/IP protocol.

OS Fingerprinting is still a new technology and additional capabilities are being generated. In the arms race of network security, concealing the OS Fingerprint of a machine should be one of the ways to enhance the security of a network system.

B. FUTURE RESEARCH

This effort showed one example of Network Fingerprint modification. One OS Fingerprint was modified to match another. It may be possible to create a “Generic OS,” that can be configurable on all aspects of its OS Fingerprint dynamically and can be configured to match any other known OS Fingerprint. By designing OS Fingerprint control into the OS, this process of changing a fingerprint could require only a change to a configuration file. By modifying an OS so its fingerprint can be changed would allow it to react to the types of attacks detected. An OS with this capability may be a good candidate for creating a “Honey Pot” or attractive trap to lure potential attackers and capture their activities.

Other future work would move beyond OS Fingerprints and into data and file fingerprints. As files become larger and more complex, more and more data becomes included. Applying the model used to originally create an OS Fingerprint could be applied to other applications, such as database or file fingerprinting. Many files already carry data on when they were created, with what application and by whom, but it may also be possible to examine the internal structure of the file and identify non-published traits such as what computer the file was modified on, what OS was used or user rights and permissions were available. Detecting the fingerprints left by users, applications and OSes on data files or databases may provide additional data. One example of this approach is how Nmap scans for banners of network services to identify the application type and version. While banners can be changed or removed, the actual format and values in the data may yet reveal information.

Fingerprinting should also apply to different layers of the communication system. As we move into a more wireless world, new technologies such as infrared, cell phones, PDA, blackberries, pagers, blue tooth, cordless phones and wireless network devices add an additional layer upon the TCP/IP protocol. These devices use various protocols to establish the wireless connection and then use TCP/IP or other protocols to transfer data. As these devices become more common they will also become more likely targeted for attacks. Scanning the wireless airway for digital fingerprints of wireless devices may allow attackers to determine the specific device and version of a target platform.

LIST OF REFERENCES

1. Lerida, J. L.; Grackzy, S. M.; Vina, A.; Andujar, J. M., Detecting security vulnerabilities in remote TCP/IP networks: an approach using security scanners. In Proc. 33rd Annual Int. Carnahan Conf. on Security Technology, IEEE (Oct. 1999), pp. 446-460.

2. Viega, J. and Voas, J. The pros and cons of Unix and Windows security policies. IT Professional, Vol. 2, Issue 5 (September/October 2000), pp. 40-47.

3. CERT Coordination Center (CERT/CC), Software Engineering Institute, Carnegie Mellon University (http://www.cert.org/), accessed June 10, 2005.

4. Millican, Andy, "Network Reconnaissance – Detection and Prevention," January 23, 2003, 13 pages, SANS Institute.

5. Jiang, W.-H., Li W.-H., and Du, J. The application of ICMP protocol in network scanning. In Proc. 4th Int. Conf. on Parallel and Distributed Computing, Applications and Technologies, IEEE (August 2003), pp. 904- 906.

6. Ofir Arkin, "ICMP usage in Scanning – The Complete Know How Version 3.0," June 2001, pages 218, WEBSITE: http://www.sys-security.com/archive/papers/ICMP_Scanning_v3.0.pdf, accessed June 10, 2005.

7. Conti, G. and Kulsoom Abdullah, K., Passive visual fingerprinting of network attack tools. In Proc. Workshop on Visualization and Data Mining for Computer Security, ACM (2004), pp. 45-45.

8. Glaser, Thomas, "Intrusion Detection FAQ, TCP/IP Stack Fingerprinting Princ.," October 25, 2000, 8 pages, WEBSITE: http://www.sans.org/resources/idfaq/tcp_fingerprinting.php, accessed June 10, 2005.

9. Beardsley, Tod, "Ring out the old, RING in the New: OS Fingerprinting through RTOs," May 8, 2002, 7 pages, WEBSITE: http://www.planb-security.net/wp/ring.html, accessed June 10, 2005.

10. Handley, Mark, et. al., "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," August 2001, 17 pages, 10th USENIX Security Symposium.

11. Malan, Robert, et. al., "Transport and Application Protocol Scrubbing," 2000, 10 pages, INFOCOM.

12. Watson, D., Smart, M., Malan, G. R., and Jahanian, F., Protocol scrubbing: network security through transparent flow modification. IEEE/ACM Trans. Netw., Vol. 12, No. 2 (2004), pp. 261-273.

13. FYDOR, “Remote OS detection via TCP/IP Stack FingerPrinting,” October 18, 1998, 11 pages, WEBSITE: http://www.insecure.org/nmap/nmap-fingerprinting-article.txt, accessed June 10, 2005.

14. Zalewski, Michal, “the new p0f: 2.0.5,” pages 3, WEBSITE: http://lcamtuf.coredump.cx/p0f.shtml, accessed June 10, 2005.

15. Beck, R. Passive-aggressive resistance: OS Fingerprint evasion. Linux J., Issue 89 (2001).

16. Hsu, F.-H. and Chiueh, T.-C. CTCP: A transparent centralized TCP/IP architecture for network security. In Proc. 20th Annual Computer Security Applications Conf., IEEE (December 2004), pp. 335-344.

17. Smart, Matthew, et. al., “Defeating TCP/IP Stack Fingerprint,” August 2000, 11 pages, Proceedings of the 9th USENIX Security Symposium.

18. Berrueta, David, “A practical approach for defeating Nmap OS-Fingerprinting,” November 2002, pages 22, WEBSITE: http://voodoo.somoslopeor.com/papers/nmap.html, accessed June 10, 2005.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia

2. Dudley Knox Library
Naval Postgraduate School
Monterey, California

3. Prof. Bret Michael
Naval Postgraduate School
Monterey, CA

4. Prof. Man-Tak Shing
Naval Postgraduate School
Monterey, California

5. Rey Yturralde
SPAWAR System Center San Diego
San Diego, California

6. Doug Lange
SPAWAR System Center San Diego
San Diego, California

7. Charles Hicks
SPAWAR System Center San Diego
San Diego, California

8. SPAWAR Technical Library
SPAWAR System Center San Diego
San Diego, California

9. Aaron Judd
SPAWAR System Center San Diego
San Diego, California